

12 Starting DQMH Modules

Launching a module

There are two ways of launching a module:

Internally

- By clicking the *Run* arrow of the module's `Main.vi`.
- The module is the top level of the execution and needs to take care of any resources, errors etc by itself.

Externally

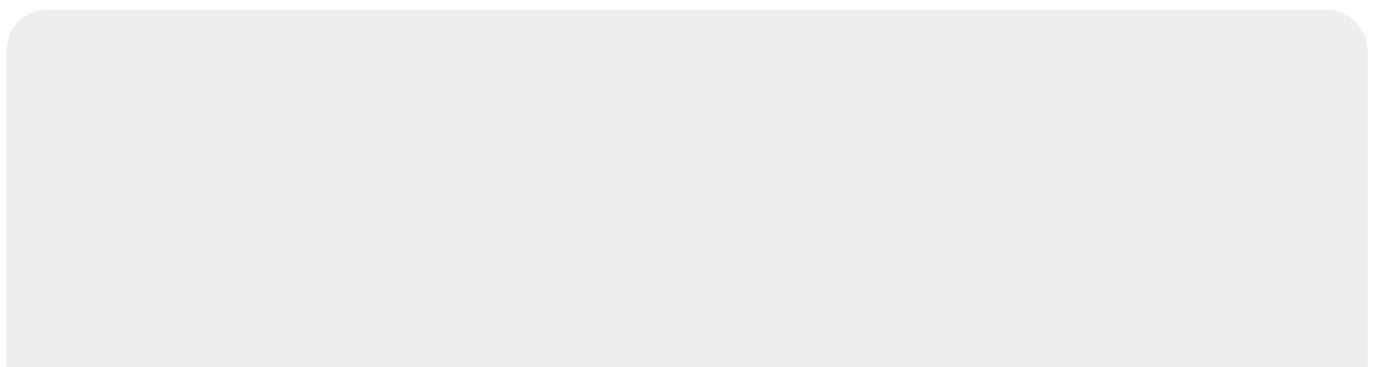
- By using the module's `Start Module.vi` and `Synchronize Module Events.vi` API functions.
- The module is not at the top level of the execution as it is being started by another VI which might or might not be a DQMH module itself. If the VI starting the module is a DQMH module itself, we refer to the module being started as a **child module** and the module starting it as the **parent module**.

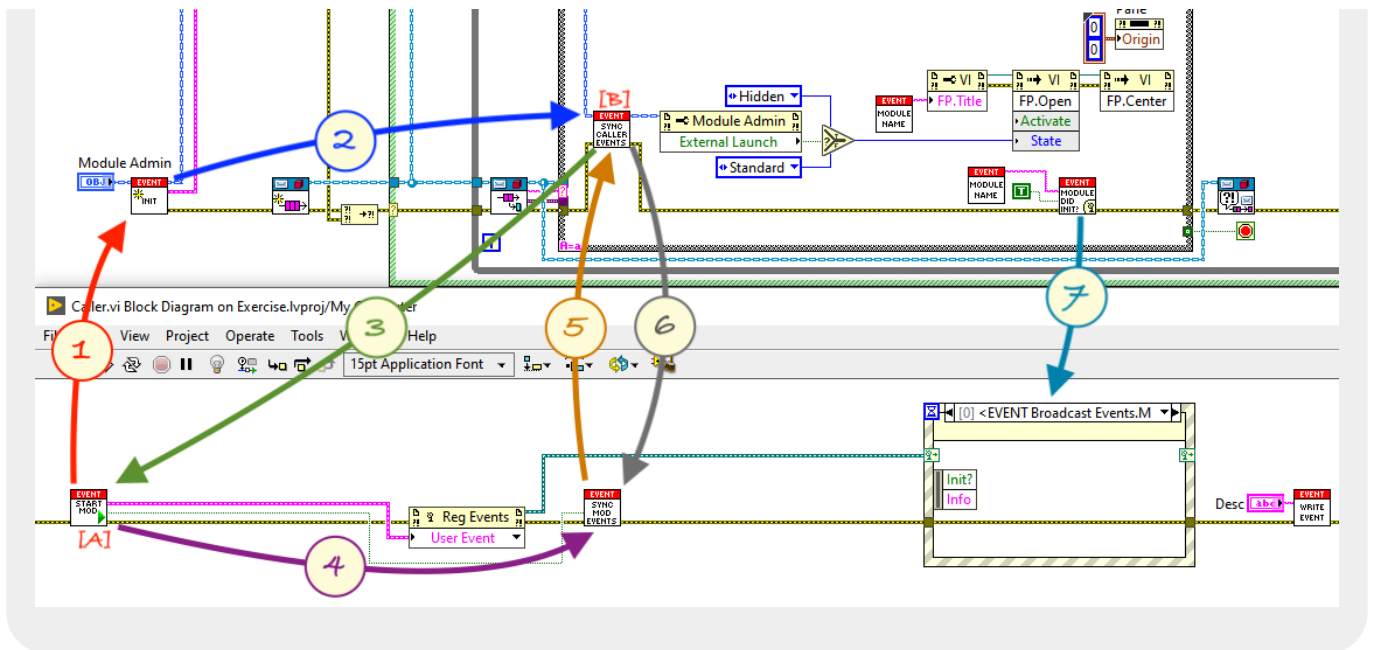
The following sections discuss aspects and ways of launching modules externally.

Startup / Synchronization

The DQMH Framework helps developers write reliable and robust code by taking care of various aspects like reference lifetime and synchronization of actions. When starting a module, the module itself creates all required resources like event references to own them and dictate their lifetime.

To ensure that the code starting the module receives any resources only after they've been created, and to make sure that the code only continues execution once the module is ready to receive requests, the DQMH Framework provides the process described in this section.





Process

1. Start and Obtain Broadcasts

- The calling VI executes `Start Module.vi`
- `Start Module.vi`...
 - uses *Start Asynchronous Call* to run the module's `Main.vi`
 - continues to `Wait` on `Module Sync.vi`, which waits until a second task arrives at the "Module Sync" rendezvous **[A]**

[A] `Start Module.vi` needs to wait until the broadcast event references have been created by the module.

- `Main.vi`
 - executes `Init Module.vi`, which creates both the module's broadcast event references and the request event references

2. Continue to MHL

- `Main.vi` continues to start Event Handling Loop (EHL), Message Handling Loop (MHL), and any Helper Loops (HL; optional)

3. Sync with Caller

- `Main.vi` enters the "Initialize" case of the MHL
 - executes `Synchronize Caller Events.vi`
 - `Wait on Module Sync.vi` adds the second task to the "Module Sync" rendezvous
 - `Start Module.vi` now continues execution
- 2. `Wait on Event Sync.vi` waits until a second task arrives at the

“Event Sync” rendezvous **[B]**

[B] The module needs to wait until the calling code has registered for the broadcast events.

- Start Module.vi returns the Broadcast event references

4. Register for Broadcasts

- The calling VI registers for the broadcast events
 - The caller will receive any broadcast sent from this point (creation of the registration) in time onwards

5. Sync with Main.vi

- The calling VI executes Synchronize Module Events.vi
 - Wait on Event Sync.vi adds the second task to the “Event Sync” rendezvous
 - In the Main.vi's “Initialize” MHL case, Synchronize Caller Events.vi now continues execution
- 2. Both rendezvous references are now closed.

The caller is now finished with starting the child module

6. Sync with Caller

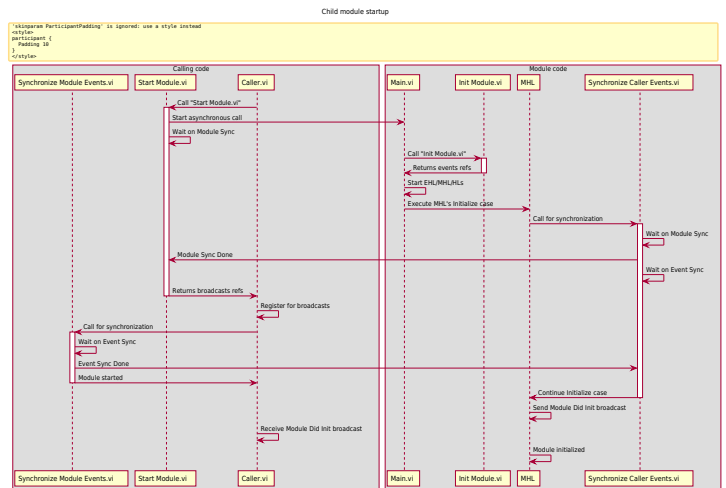
- In the “Initialize” MHL case of the Main.vi, Synchronize Caller Events.vi now returns and the rest of the code in that case is executed.

7. Module Did Init

- Once the code in the “Initialize” MHL case has finished executing, the Module Did Init.vi broadcast is sent.

The child module is now finished with starting

- The Caller can choose to register to this broadcast for added synchronization.



Implications

Any rendezvous functions mentioned above will wait for the module timeout defined in `Module Timeout-constant.vi` (default = 5s) before returning an error.

It becomes apparent that **2. Continue to MHL** in the schema shown above is the critical path. If any code is placed before the MHL starts (i.e. before `Synchronize Caller Events.vi` inside the “Initialize” case) which takes longer to execute than the module timeout, the startup process of the module will fail.

Caveat



Be careful when adding code outside of the EHL/MHL/HLs, especially code that needs to execute before entering the EHL/MHL/HLs and might therefore delay entering the “Initialize” MHL case. If it takes too long to reach the “Initialize” case, the module synchronisation will time out and fail, and the `Start Module.vi` will throw error 403683 (“Module was unable to synchronise events”).

Starting Child Modules

Make sure to also browse our [HSE Coding Conventions on Parent/Child Modules](#).

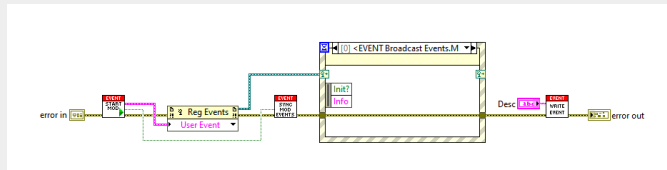


This section describes ways of starting child modules. It lists different options and explains shortcomings/challenges.

At HSE, the only allowed way is **4.2 In the MHL - Variant 2**.

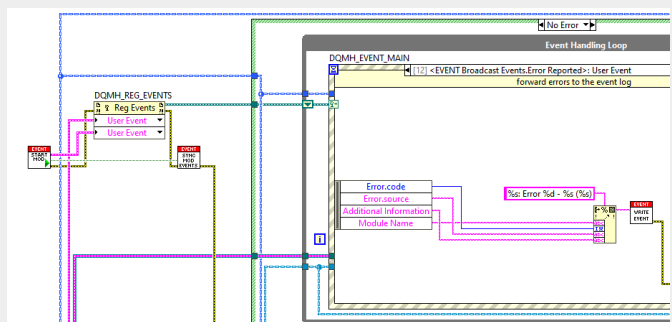
1. From a Simple VI

- It is not from another DQMH Module. One example is the API Tester.



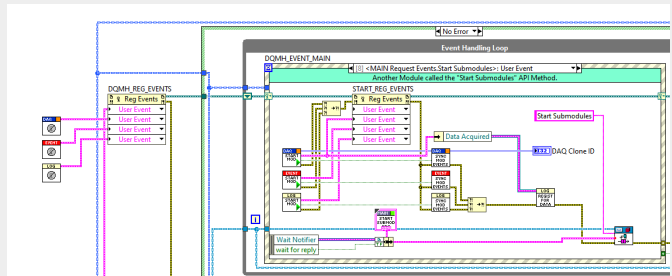
2. At Module Start

- Child module is started automatically when starting the parent module
- Adding code outside of EHL/MHL (see [Caveat above](#))
- Risk of timeouts → not recommended by HSE



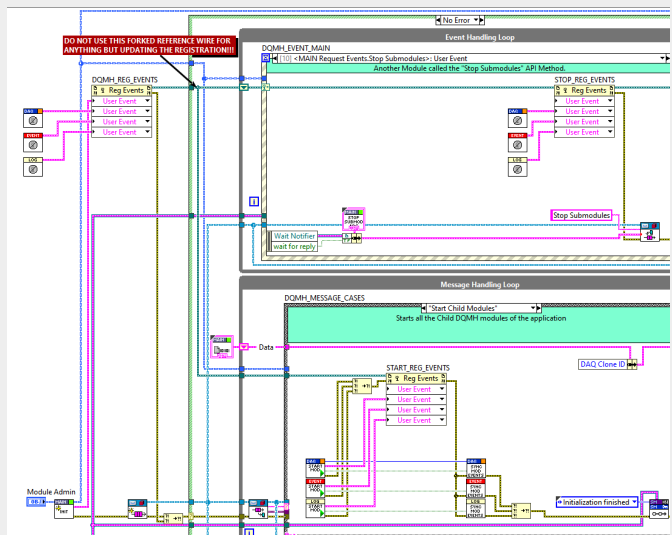
3. In the EHL

- Child module is started via a request [1] by adding code to the EHL
- EHL (producer) now contains code that should ideally be placed in MHL (consumer)
- Code potentially blocking the EHL → not recommended by HSE



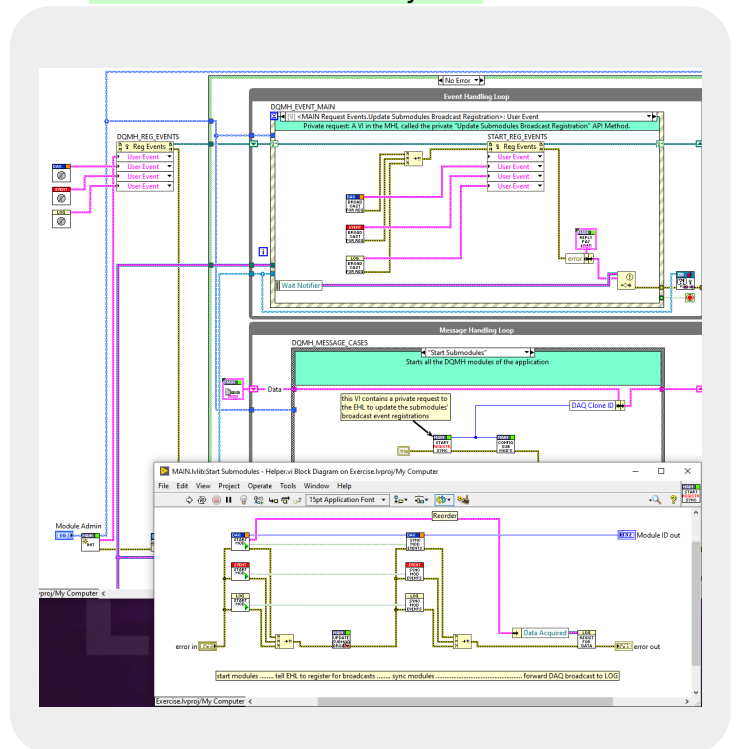
4.1 In the MHL - Variant 1

- Child module is started inside MHL, can be triggered via request or simple Queue message [1]
- Forking the registration reference wire so we can update the event registration of the EHL in the MHL
- Potential confusion about forked registration wire - not recommended by HSE



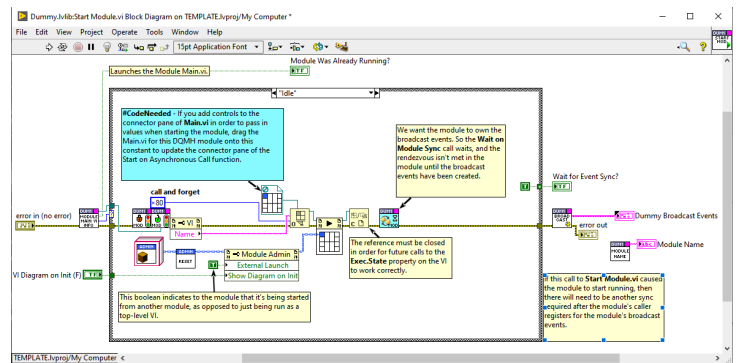
4.2 In the MHL - Variant 2

- Child module is started inside MHL, can be triggered via request or simple Queue message [1]
- Using a private request and wait for reply to let the EHL register while executing the MHL case
- More implementation work but little technical debt → recommended by HSE



Startup Parameters

You can pass in parameters directly when starting a DQMH module. This is done by adding controls to the connector pane of Main.vi and then wiring those in Start Module.vi. A #CodeNeeded comment on the block diagram explains the implementation:



There are two distinct results this approach allows for that cannot be reached otherwise:

1. Have that input parameter available as soon as the `Main.vi` starts executing (ie even before the "Initialize" case etc.)
2. Force the user of the module to provide a value at start time (whereas you cannot force a user to send requests in a certain order)

The second argument is not a strong one, as a user can still provide a "wrong" value so there will always be the need for proper error handling, regardless of how parameters are provided.



HSE's rule of thumb is to always pass parameters via requests unless there is a strong argument for hard-wiring them like this. We feel it's safer to go with the standard way of doing things (ie requests) as this will make it easier for others to read the code. Also, passing parameters via request gives control over when to send or resend that data.

When to start

The concept of placing the code for starting modules inside EHL/MHL and having a request or Queue message trigger the starting of child modules gives control and flexibility over when and from where modules are actually (re)started. It does beg the question, though, of where and when to actually send the request or message triggering the start.

- HSE modules with the need for automated sequential steps usually feature a [State Machine](#).
 - Our [HSE Application Template](#) calls the “Configure” request automatically for any module it starts, which is another potential place for sending such a request (usually in combination with a state machine, though)
-

Better Practices



Our better practices section on [Coding Conventions](#) contains a page on DQMH®:
[DQMH Coding Conventions](#)

From:
<https://dokuwiki.hampel-soft.com/> - HAMPEL SOFTWARE ENGINEERING

Permanent link:
<https://dokuwiki.hampel-soft.com/kb/labview-frameworks/dqmh/starting-modules?rev=1775721219>

Last update: 2026/04/09 07:53

